Performance optimisation at UT

Victor Eijkhout

ICL, University of Tennessee, Knoxville

TOPS scidac
25/26 January 2002
Livermore CA

Topics of optimisation

- Matrix structure detection

Topics of optimisation

- Matrix structure detection
- A^2x kernel

Topics of optimisation

- Matrix structure detection
- A^2x kernel
- Multigrid smoothers

Matrix structure detection

Observations:

- Certain parallel preconditioners imply physical domain partitioning (Block Jacobi, but not multicolour ILU)

Matrix structure detection

Observations:

- Certain parallel preconditioners imply physical domain partitioning (Block Jacobi, but not multicolour ILU)
- ’Natural’ domain partitioning often acknowledges partitioning of the physics.

Matrix structure detection

Observations:

- Certain parallel preconditioners imply physical domain partitioning (Block Jacobi, but not multicolour ILU)
- ’Natural’ domain partitioning often acknowledges partitioning of the physics.

$\Rightarrow$ Let partioning for parallel processing acknowledge the same structure.

Structure test

Re-engineering of level sets:

Structure test

Re-engineering of level sets:

- first point connects to first point of next set

Structure test

Re-engineering of level sets:

- first point connects to first point of next set
- last point connects to last point of next set

Structure test

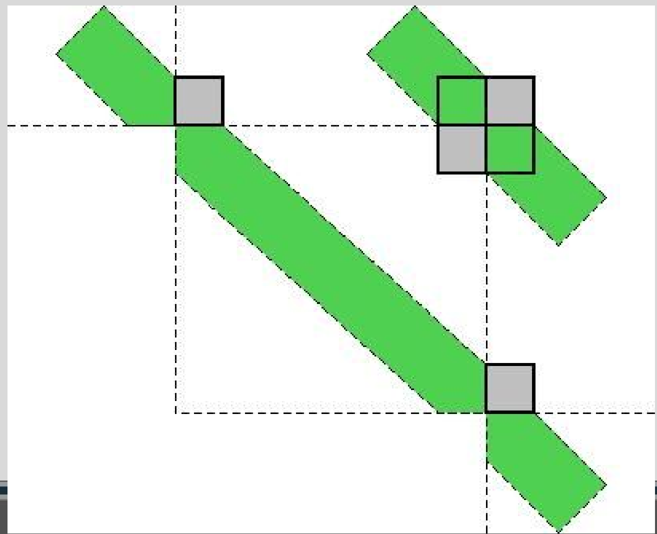
Re-engineering of level sets:

- first point connects to first point of next set
- last point connects to last point of next set
- first point does not connect to last point of another set

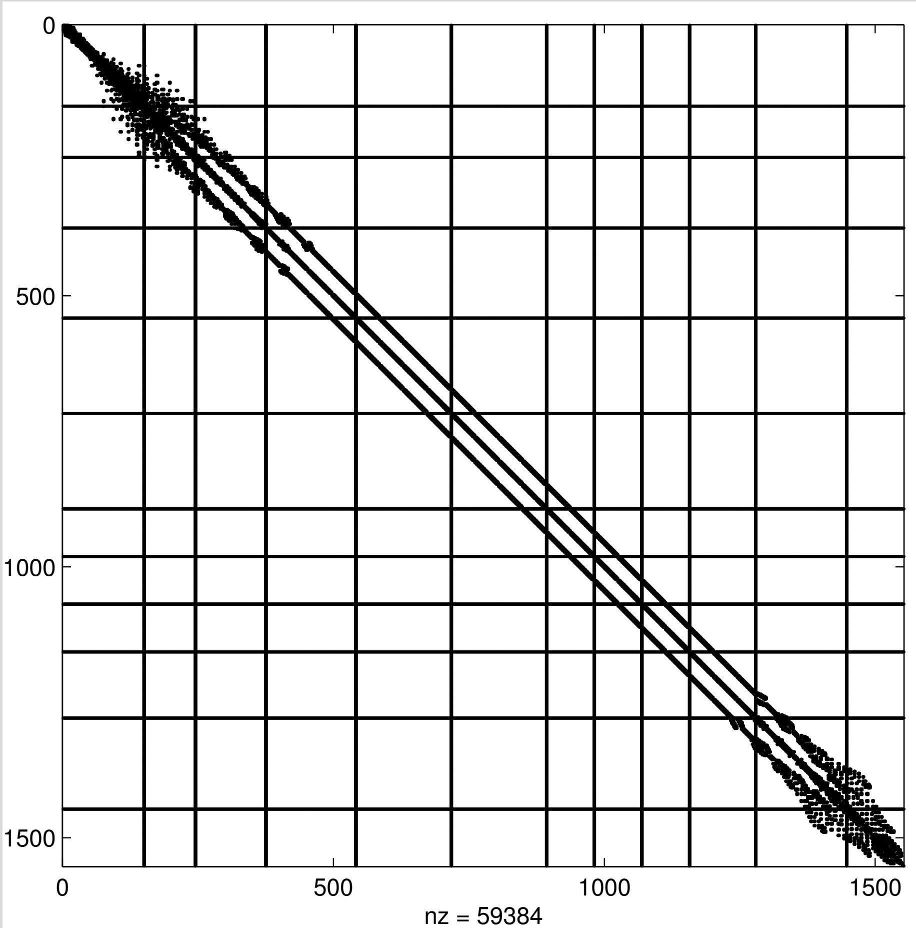
Structure test

Re-engineering of level sets:

- first point connects to first point of next set
- last point connects to last point of next set
- first point does not connect to last point of another set



Structure detection example



State of the work

Parallel implementation in Petsc

can be made more efficient by adding new Petsc primitives

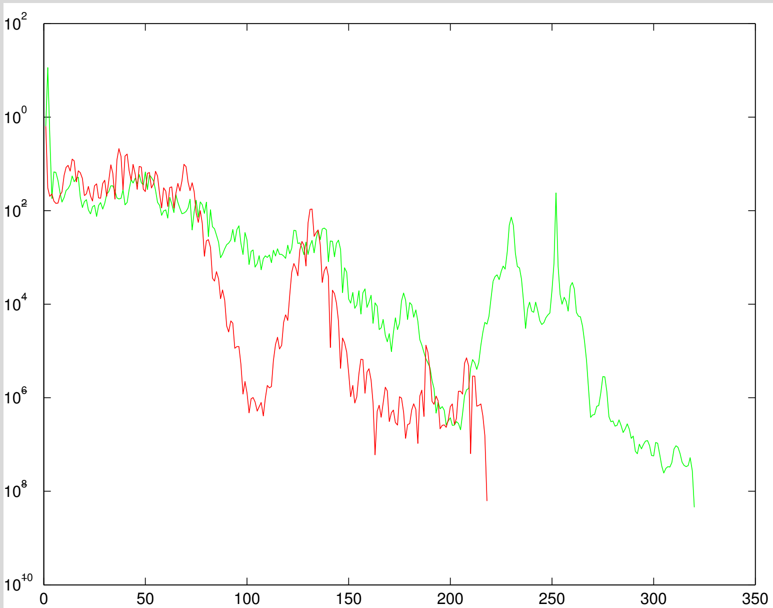
State of the work

Parallel implementation in Petsc

can be made more efficient by adding new Petsc primitives

Small number of tests done

> *more to be done*
>
> *comparison with Chaco &c.*



A^2x kernel

(work for TSI scidac)

- Matrix is direct product of block diagonal and tridiagonal

A^2x kernel

(work for TSI scidac)

- Matrix is direct product of block diagonal and tridiagonal
- ADI preconditioner $\Rightarrow$ solve many small dense systems

A^2x kernel

(work for TSI scidac)

- Matrix is direct product of block diagonal and tridiagonal
- ADI preconditioner $\Rightarrow$ solve many small dense systems
- Solution of small (30–3k) dense systems by iterative method.
  well-conditioned, so low number of iterations.

Iterative solution

- Formulate as left-preconditioned method

$$A = (D - E) \equiv D(I - N), \quad M^{-1} = (I + N)D^{-1}$$

$$\Rightarrow \quad M^{-1}A = (I - N^2)$$

Iterative solution

- Formulate as left-preconditioned method

$$A = (D - E) \equiv D(I - N), \quad M^{-1} = (I + N)D^{-1}$$

$$\Rightarrow \quad M^{-1}A = (I - N^2)$$

- depends on efficiency of $y = N^2 x$
  can we beat twice-`gemv`?

Efficiency of A^2x kernel

Why we can beat twice-`gemv`:

Efficiency of A^2x kernel

Why we can beat twice-`gemv`:

- Reuse of matrix

Efficiency of A^2x kernel

Why we can beat twice-`gemv`:

- Reuse of matrix
- Possible elimination of intermediate result

Efficiency of A^2x kernel

Why we can beat twice-`gemv`:

- Reuse of matrix
- Possible elimination of intermediate result
- Atlas gemv is optimised for out-of-cache; we possibly operate in-cache

Recursive approach to out-of-cache case

Let

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

then

$$\begin{aligned} y_1 &= A_{11}(A_{11}x_1 + A_{12}x_2) + A_{12}(A_{21}x_1 + A_{22}x_2) \\ y_2 &= A_{21}(A_{11}x_1 + A_{12}x_2) + A_{22}(A_{21}x_1 + A_{22}x_2) \end{aligned}$$

Recursive approach continued

Introduce $t = Ax$ and localise application of A_{11}, A_{22}:

$$
\begin{aligned}
\left.\begin{matrix} t_1 \\ t_2 \end{matrix}\right\} &= \left\{\begin{matrix} A_{12}x_2 \\ A_{21}x_1 \end{matrix}\right. \\
y_1 &= A_{11}\hat{t}_1, \quad \hat{t}_1 = A_{11}x_1 + t_1 \\
y_2 &= A_{22}\hat{t}_2, \quad \hat{t}_2 = A_{22}x_2 + t_2 \\
\left.\begin{matrix} y_1 \\ y_2 \end{matrix}\right\} &+= \left\{\begin{matrix} A_{12}\hat{t}_2 \\ A_{21}\hat{t}_1 \end{matrix}\right.
\end{aligned}
$$

New kernel for recursive A^2x

Instructions involving reuse:

$$
\begin{aligned}
y_1 &= A_{11}\hat{t}_1, \quad \hat{t}_1 = A_{11}x_1 + t_1 \\
y_2 &= A_{22}\hat{t}_2, \quad \hat{t}_2 = A_{22}x_2 + t_2
\end{aligned}
$$

New kernel for recursive A^2x

Instructions involving reuse:

$$\begin{aligned} y_1 &= A_{11}\hat{t}_1, \quad \hat{t}_1 = A_{11}x_1 + t_1 \\ y_2 &= A_{22}\hat{t}_2, \quad \hat{t}_2 = A_{22}x_2 + t_2 \end{aligned}$$

Double matrix vector product

`[y,s]=m2v(A,t,x)`:

$$s = Ax + t, \qquad y = As$$

Example

On ev6:

	32	68	92	128
blas	356	415	395	389
unroll4	439	450	432	322

Example

On ev6:

	32	68	92	128
blas	356	415	395	389
unroll4	439	450	432	322

		64	96	
unroll4		386	399	

Multigrid smoothers

- Scalar optimisation (Atlas techniques); with Jun Ding.

Multigrid smoothers

- Scalar optimisation (Atlas techniques); with Jun Ding.
- Mathematical optimisation

  construct CG iterates from GS iterates

  does this pay?

  other spectrum-adaptive method?

Summary

- Optimisation of dense and sparse kernels
- Optimisation: uni-processor and distributed
- Optimisation through Intelligent adaptation